CS634

DATA MINING PROJECT

vm567

Creating a USPTO application to calculate the Patentability score of different datasets.

United States Patent and Trademark Office application predicts a patentability score of a dataset based on the information provided in the dataset.

Source code :

```
from pprint import pprint

from datasets import load_dataset

from transformers import AutoTokenizer,pipeline

from torch.utils.data import DataLoader

import streamlit as st


import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, recall_score, precision_score,
f1_score
import torch
from transformers import TrainingArguments, Trainer
from transformers import BertTokenizer,
BertForSequenceClassification,AutoTokenizer,AutoModelForSequenceClassification
from transformers import DistilBertForSequenceClassification,
DistilBertTokenizer, DistilBertConfig
```

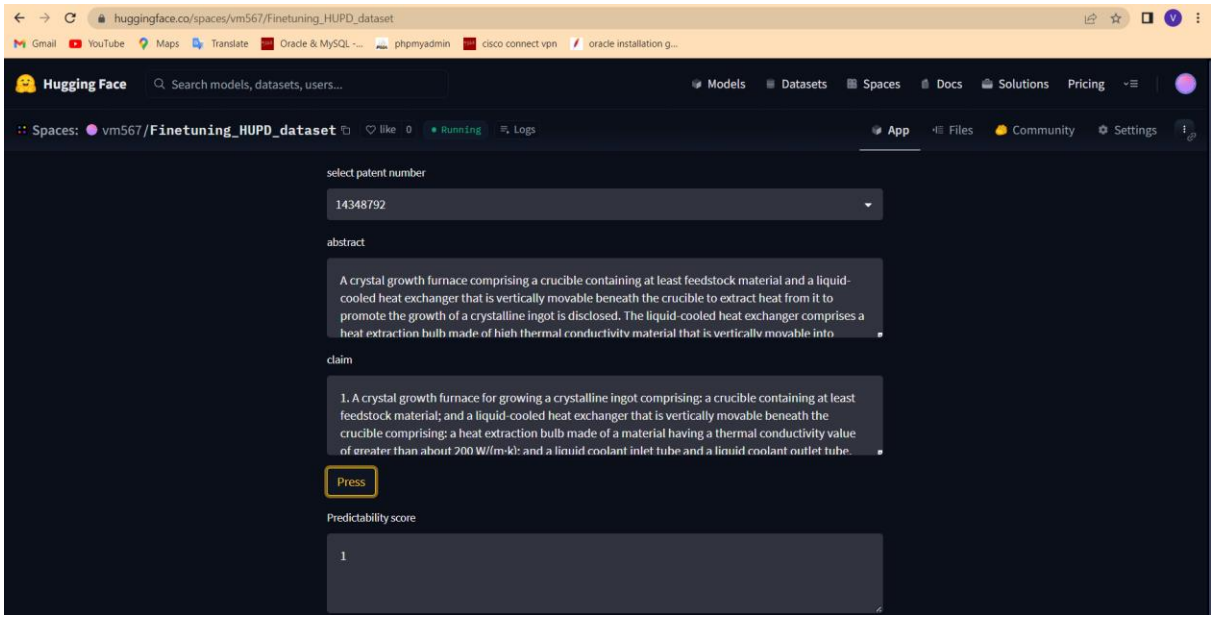
```
dataset_dict = load_dataset('HUPD/hupd',
    name='sample',
    data_files="https://huggingface.co/datasets/HUPD/hupd/blob/main/hupd_metad
ata_2022-02-22.feather",
    icpr_label=None,
    train_filing_start_date='2016-01-01',
    train_filing_end_date='2016-01-21',
    val_filing_start_date='2016-01-22',
    val_filing_end_date='2016-01-31',
)

print('Loading is done!')

print(dataset_dict)

print(f'Train dataset size: {dataset_dict["train"].shape}')
print(f'Validation dataset size: {dataset_dict["validation"].shape}')

decision_to_str = {'REJECTED': 0, 'ACCEPTED': 1, 'PENDING': 2, 'CONT-
REJECTED': 3, 'CONT-ACCEPTED': 4, 'CONT-PENDING': 5}
def map_decision_to_string(example):
    return {'decision': decision_to_str[example['decision']]}




# Re-labeling/mapping.
train_set = dataset_dict['train'].map(map_decision_to_string)
val_set = dataset_dict['validation'].map(map_decision_to_string)

print(train_set)







train_set_reduced =
train_set.remove_columns(['title','background','summary','description','cpc_la
bel','ipc_label','filing_date','patent_issue_date','date_published','examiner_
id'])

val_set_reduced =
val_set.remove_columns(['title','background','summary','description','cpc_labe
```

```
l','ipc_label','filing_date','patent_issue_date','date_published','examiner_id
'])

print(train_set_reduced)

train_set_reduced = train_set_reduced.filter(lambda row: row["decision"] < 2)
val_set_reduced = val_set_reduced.filter(lambda row: row["decision"] < 2)


print(train_set_reduced['decision'])
print(type(train_set_reduced))


# for the app

train_df_app=train_set_reduced.data.to_pandas()
val_set_app =val_set_reduced.data.to_pandas()


option = st.selectbox('select patent number',train_df_app['patent_number'])
idx_pos = list(np.where(train_df_app['patent_number'] == option))

abstract_text = train_df_app['abstract'].iloc[idx_pos[0][0]]
claim_text = train_df_app['claims'].iloc[idx_pos[0][0]]
decision_text = train_df_app['decision'].iloc[idx_pos[0][0]]


st.text_area("abstract",abstract_text)
st.text_area("claim",claim_text)


if st.button("Press"):
    st.text_area("Predictability score",decision_text)


# --------------------------------




tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('bert-base-
uncased',num_labels=2)

for row in train_set_reduced:
```

```
  row["abstract"] = tokenizer(row["abstract"], padding=True, truncation=True,
                              max_length=512)
  row["claims"] = tokenizer(row["claims"], padding=True, truncation=True,
                              max_length=512)

X_train_col = train_set_reduced.remove_columns(['decision'])
Y_train_col =
train_set_reduced.remove_columns(['patent_number','abstract','claims'])



X_train, X_test, y_train, y_test = train_test_split(X_train_col, Y_train_col,
test_size=0.2)

class Dataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels=None):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in
self.encodings.items()}
        if self.labels:
            item["labels"] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.encodings["input_ids"])

X_train_encodings = tokenizer(list(X_train),padding = True, truncation =
True,max_length=512)
X_test_encodings = tokenizer(list(X_test),padding = True, truncation =
True,max_length=512)

Y_train_encodings = tokenizer(list(y_train),padding = True, truncation =
True,max_length=512)
y_test_encodings = tokenizer(list(y_test),padding = True, truncation =
True,max_length=512)

print(X_train_encodings.items())

print(Y_train_encodings)

x_train_dataset = Dataset(X_train_encodings,Y_train_encodings)
X_test_dataset = Dataset(X_test_encodings,y_test_encodings)
```

```
print(x_train_dataset)

print(type(X_train_encodings))
print(type(x_train_dataset))

def compute_metrics(p):
    print(type(p))
    pred, labels = p
    pred = np.argmax(pred, axis=1)

    accuracy = accuracy_score(y_true=labels, y_pred=pred)
    recall = recall_score(y_true=labels, y_pred=pred)
    precision = precision_score(y_true=labels, y_pred=pred)
    f1 = f1_score(y_true=labels, y_pred=pred)

    return {"accuracy": accuracy, "precision": precision, "recall": recall,
"f1": f1}

# Define Trainer
args = TrainingArguments(
    output_dir="output",
    num_train_epochs=1,
    per_device_train_batch_size=8

)
trainer = Trainer(
    model=model,
    args=args,
    train_dataset=x_train_dataset,
    eval_dataset=X_test_dataset,
    compute_metrics=compute_metrics
)

# trainer.train()
```

OUTPUT:


huggingface.co/spaces/vm567/Finetuning_HUPD_dataset
Gmail
YouTube
Maps
Translate
Oracle & MySQL -...
phpmyadmin
cisco connect vpn
oracle installation g...
Hugging Face
Search models, datasets, users...
Models
Datasets
Spaces
Docs
Solutions
Pricing
Spaces: vm567/Finetuning_HUPD_dataset
like 0
Running
Logs
App
Files
Community
Settings
select patent number
14348792
abstract
A crystal growth furnace comprising a crucible containing at least feedstock material and a liquid-cooled heat exchanger that is vertically movable beneath the crucible to extract heat from it to promote the growth of a crystalline ingot is disclosed. The liquid-cooled heat exchanger comprises a heat extraction bulb made of high thermal conductivity material that is vertically movable into
claim
1. A crystal growth furnace for growing a crystalline ingot comprising: a crucible containing at least feedstock material; and a liquid-cooled heat exchanger that is vertically movable beneath the crucible comprising: a heat extraction bulb made of a material having a thermal conductivity value of greater than about 200 W/(m-k); and a liquid coolant inlet tube and a liquid coolant outlet tube.
Press
Predictability score
1